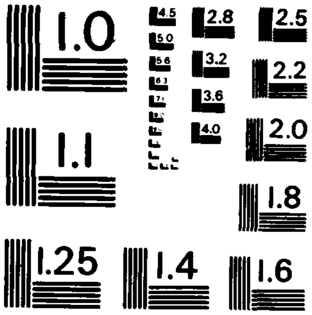
AD-A133 905 THE SYNTACTIC THEORY OF BELIEF AND KNOWLEDGE(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA A R HAAS SEP 83 BBN-5368 N00014-77-C-0378 1/1

UNCLASSIFIED F/G 6/4 NL

END
DATE
FILMED
9 8
DTIC


1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

Bolt Beranek and Newman Inc.

Report No. 5368

The Syntactic Theory of Belief and Knowledge

Andrew R. Haas

September 1983

Prepared for:
Defense Advanced Research Projects Agency

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
BBN Report No. 5368

2. GOVT ACCESSION NO.
AD-A133906

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
THE SYNTACTIC THEORY OF BELIEF AND KNOWLEDGE

5. TYPE OF REPORT & PERIOD COVERED
Technical Report

6. PERFORMING ORG. REPORT NUMBER
BBN Report No. 5368

7. AUTHOR(s)
Andrew R. Haas

8. CONTRACT OR GRANT NUMBER(s)
N00014-77-C-0378
N00014-78-C-0164

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Bolt Beranek and Newman Inc.
10 Moulton Street
Cambridge, MA 02238

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Office of Naval Research
Department of the Navy
Arlington, VA 22217

12. REPORT DATE
September 1983

13. NUMBER OF PAGES
81

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES
This research was supported in part by the National Science Foundation, the Office of Naval Research, and the Defense Research Projects Agency.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Knowledge Representation, Belief, Propositional Attitudes, Intropsection, Quantifying In

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
If we assume that beliefs are sentences of first-order logic stored in an agent's head, we can build a simple and intuitively clear formalism for reasoning about beliefs. I apply this formalism to the standard logical problems about belief, and use it to describe the connections between belief and planning.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Report No. 5368

THE SYNTACTIC THEORY OF BELIEF AND KNOWLEDGE

Andrew R. Haas

September 1983

Prepared by:

Bolt Beranek and Newman Inc.
10 Moulton Street
Cambridge, Massachusetts 02238

Prepared for:

Defense Advanced Research Projects Agency

The work described herein was supported in part by the National Science Foundation IST-8012418, the Office of Naval Research N00014-8-C-0197, and the Defense Advanced Research Projects Agency, N00014-78-C-0164 and N00014-77-C-0378. The views and conclusions in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies or the U.S. Government.

TABLE OF CONTENTS

ABSTRACT

If we assume that beliefs are sentences of first-order logic stored in an agent's head, we can build a simple and intuitively clear formalism for reasoning about beliefs. I apply this formalism to the standard logical problems about belief, and use it to describe the connections between belief and planning.

1. Belief and Knowledge in Artificial Intelligence

1.1 Representation and Search

Artificial Intelligence programs must have common-sense knowledge. This includes knowledge about beliefs and knowledge, A program must be able to understand that Bill believes Mary's phone number is 5766, or that John knows the name of every person in the department. If a program is supposed to understand these facts, it should be able to make the right inferences from them. If Bill knows that Mary's phone number is 5766, he knows what Mary's phone number is. If a program thinks that Bill knows that Mary's number is 5766, the program should be able to infer that Bill knows what Mary's number is. If we have a knowledge representation that can represent facts about beliefs and knowledge, and an adequate set of inference rules, we have taken the first step in building a program that can reason about beliefs and knowledge. The next step is to devise a search strategy: an algorithm that decides which inference rules to apply to which expressions to solve a problem,

This paper ~~is~~ about the first step. ~~It~~ proposes a representation and inference rules for reasoning about belief and knowledge. Section 1.2 presents examples of sound and unsound inferences about belief and knowledge. The problem is to allow all the sound inferences and rule out the unsound ones. The best treatment to date is Moore's [13], and I discuss his successes and failures. Section 2 presents the syntactic theory of belief and shows how to formalize it. Section 3 is the core of the paper~~:~~ is a series of examples of representation and inference in the formal system. These examples describe the processes that create, store and use beliefs and knowledge. Perception, introspection, memory, inference and planning are all considered. Finally there is an appendix with proofs that the formalism works as claimed.

Recent work has made great improvement in AI theories of belief and knowledge, but they still have serious problems. For example, Moore's theory predicts that agents always know every logical consequence of their knowledge. This theory tries to solve the problems by formalizing familiar ideas from computer science. For example, it says that sentences stored in an agent's memory represent his beliefs. It gives better answers to several questions: When does an agent need knowledge to perform an action? What can an agent infer from his beliefs? What must you know about an object in order to know what that object is?

1.2 Some Inferences About Belief and Knowledge

Let us consider some examples that show why reasoning about beliefs is hard. For one thing, the familiar rule of substitution of equals does not apply when one of the equals appears inside the scope of the verb "believe". For example, the following inference is not correct.

John believes that Mary's phone number is 444-1212.
Bill's phone number is Mary's phone number.

John believes that Bill's phone number is 444-1212.

It is easy enough to forbid the substitution of equals when one of the equals appears inside the scope of "believe", but this is not very satisfying. One would like an explanation of why substitution of equals does not apply.

The following inferences are correct:

John knows that snow is white.

John believes that snow is white.

John knows that snow is white.

Snow is white.

That is, all knowledge is true belief. On the other hand, not all true beliefs are knowledge. Suppose somebody predicts that a horse will win a race when the odds are 30 to 1 against it. Sure enough, the horse wins. We might ask "How did he know the horse would win?" It would make sense to answer "He didn't know, it was just a lucky guess." That is, a true belief might not count as knowledge if there is no good reason for the belief. I will not consider this problem further. Suffice it to say that all knowledge is true belief.

The following inference is correct.

John knows that Mary's phone number is 444-1212.

John knows what Mary's phone number is.

But this one is not necessarily correct:

John knows that Mary's phone number is Bill's phone number.

John knows what Mary's phone number is.

This raises the question: when does John's knowing that X is N entail that John knows what X is? The noun phrases "444-1212" and "Bill's phone number" both denote Mary's phone number, but knowing that Mary's number is Bill's number does not count as knowing what Mary's phone number is. In some sense the phrase "Bill's phone number" does not contain enough information, but it's hard to clarify this.

Context helps to decide what knowledge about X counts as knowing what X is. Suppose that you and John are staying at a hotel in a strange city, and you go out for a walk. After a while John asks "Do you know where we are?" You realize that you're completely lost, and answer "No." Seeing a telephone you decide to call Mary and ask for directions. She answers and says "Do you know where John is? I need to talk to him right away." You answer "Yes, he's right here" and hand him the phone. When John asked if you knew where he was you said no; a moment later you answered yes to the same question.

If you had answered John's question with "Yes; we're right here", he would not have been amused. John wanted information that would help him to get back to the hotel. Mary wanted information that would help her to get in touch with John, and for that purpose "right here" was a useful description of John's location.

One clue to the problem of "knowing what" comes from the problem of "knowing how". The following inference is correct:

John knows that Mary's number is 444-1212.
John knows how to dial a telephone.

John knows how to dial Mary's number.

This one is not:

John knows that Mary's number is Bill's number.
John knows how to dial a telephone.

John knows how to dial Mary's number.

We saw that if you have the name "444-1212" for Mary's number you know what her number is, but not if you only have the name "Bill's number". Similarly, if you have the name "444-1212" for Mary's number you know how to dial the number, but not if you only have the name "Bill's number". It is tempting to connect these two facts. In any case a theory of belief and knowledge must say something about what knowledge is needed to perform actions. So the theory of belief and knowledge is connected to the theory of planning.

The problem of "knowing what" is closely related to the so-called de re statements about belief. Suppose you see John in a restaurant with a woman you don't know, and you think "That must be John's wife". Later you find she was his sister. You might say "I thought John's sister was his wife." The following inference is correct, at least in some contexts:

I thought John's sister was an accountant.

I believed the statement "John's sister is an accountant".

But the following is surely not correct in this context:

I thought John's sister was his wife.

I believed the statement "John's sister is his wife".

In this case the example seems to mean about the same as "I saw

John's sister and thought she was his wife". The speaker uses the description "John's sister" to identify the woman he took for John's wife. Such statements are called de re reports of belief or knowledge.

Truth is a crucial property of beliefs. Our theory must explain inferences like this:

John believes that gold is an element.
Everything that John believes is true.
____
Gold is an element.

If we know that someone's beliefs are true, we can infer things about the objects those beliefs refer to. We can also reason in the other direction: if an object has certain properties, then certain beliefs about it are true.

Coal is black.
John believes that coal is black.
____
John believes something true.

Common sense says that we think about objects outside our heads, and that our beliefs about them can be right or wrong.

People use their beliefs to infer new beliefs. What they infer depends on what problems they want to solve and how hard they think. For example, the following inference is very plausible.

John knows that Mary's number is 5766.

John knows that Mary's number is Bill's number.

John is trying to figure out what Bill's number is.

John will infer that Bill's number is 5766.

On the other hand, a math teacher had better not accept the following:

The students believe the Axiom of Choice.

The Axiom of Choice entails that every set can be well-ordered.

The students will infer that every set can be well-ordered.

A theory of belief ought to distinguish hard inferences from easy ones, and it ought to say that what people infer from their beliefs depends on what they try to infer.

People know about their own beliefs. They can easily answer questions like "Do you know what Mary's phone number is?". Yet we don't want to claim that people always know about all their beliefs, anymore then we want to claim that they believe everything that they could infer from their beliefs. Otherwise, we would end up with the following as a valid chain of inference:

John believes that snow is white.

John believes that John believes that snow is white.

John believes that John believes that John believes that snow is white.

John believes that John believes that John believes that John believes...

The second line is plausible enough, but the fourth line is weird, and if we continued the 500th line would be impossible to read, let alone believe. Introspection is like inference: it is something people do on purpose, and they do as much of it as they need for the problem at hand.

Many beliefs are the result of perception. People make inferences like the following:

John looked at a piece of paper with a number written on it.

John knew what number was written on the paper.

I stressed above that many beliefs arise from a deliberate effort of thinking. If we say that inference and introspection happen automatically, we get into trouble because these processes take beliefs as input and produce new beliefs as output. Therefore their output can be used as input for more introspection and inference, and if the process runs on automatically we might get an infinite set of beliefs. This problem does not occur with perception, because its input is not old beliefs, but physical events in the external world. Therefore no problem arises if we claim that perception creates new beliefs automatically. And this seems to be true. If someone sneaks up behind your back and blows a bugle in your ear, you'll notice it whether you want to or not.

1.3 The Situation Theory

Robert Moore's dissertation [13] uses a theory of belief based on Hintikka's possible worlds theory [6]. Moore had the ingenious idea of replacing Hintikka's possible worlds with the situations of McCarthy's situation calculus. Recall that the situation calculus is a technique for reasoning about actions. It introduces entities called situations, such that an object can have different properties in different situations, and at each instant of time the world is in exactly one situation. Since the properties of objects vary from situation to situation, a sentence can be true in one situation and false in another. Also, a description like "Bill's phone number" can denote different objects in different situations. One describes an action as a relation over situations. If this relation holds between situations s1 and s2, you can perform the action at any instant when the world is in situation s1, and if you do the world will be in situation s2 at the next instant. Moore dealt with knowledge only, but I will consider a natural extension of his theory to belief.

Moore proposed to represent an agent's beliefs as a set of situations, which I will call the agent's alternatives. If situation s is one of the agent's alternatives, then the agent's beliefs do not rule out the possibility that the current situation is s. In other words, for all he knows the world might be in situation s. Thus if the agent knows everything about the current situation, his set of alternatives contains only the actual situation. If he knows nothing at all his set of alternatives contains every situation. The more the agent learns, the more situations he rules out and the fewer his alternatives.

An agent believes that P if P is true in all of his alternatives. This explains at once why substitution of equals fails inside the scope of "believe". If John believes that Mary's number is 444-1212, then Mary's number is 444-1212 in all of his alternatives. If Bill's number is Mary's number, then Bill's number is Mary's number in the actual situation, and so Bill's number is 444-1212 in the actual situation. Still John's alternatives might include situations in which Bill and Mary have different numbers, and in these alternatives Bill's number is not 444-1212. So John does not necessarily believe that Bill's number is 444-1212.

This theory will also handle the first "knowing what" example. Moore says that an agent knows what X is if X is the same object in all of the agent's alternatives. That is, the agent's beliefs rule out all but one value of X. If the agent knows that Mary's number is 444-1212, then Mary's number is 444-1212 in all the agent's alternatives. Surely 444-1212 is the same number in all situations. That number is Mary's phone number in all of the agent's alternatives, so the agent knows what Mary's number is. On the other hand, suppose the agent knows only that Mary's number is Bill's number. Bill might have different phone numbers in different situations, so there need not be any one object that is Mary's number in all of the agent's alternatives. Again we get the right prediction.

Moore goes on to say that actions take arguments. For example, the action of dialing a phone number takes one argument, the number to be dialed. An agent knows how to perform an action only if he knows what the action's arguments are. Then it follows that an agent knows how to dial Mary's number if he knows that Mary's number is 444-1212, but not if he only knows that Mary's number is Bill's number.

I claim that the situation theory of belief is wrong, and that a very different approach is needed (this is also Moore's current view - see [14]). The first criticism is that it makes false predictions about "knowing what". We don't say that you know what Mary's number is if you know that her number is equal to six times thirty-one squared. Yet six times thirty-one squared is surely the same number in every situation. In this case Mary's phone number is the same number in all of the agent's alternatives, yet he still doesn't know what her phone number is. Also, according to the situation theory whether an agent knows what X is depends only on the agent's alternatives. But we have seen that it can depend also on what the agent wants to do with the knowledge. If you want to put Mary in touch with John, and you know that John is standing next to you, you claim that you know where John is. If you want to direct John back to his hotel, and you know that he is standing next to you, you must learn more before you can claim to know where he is.

Suppose the agent believes that P, and P entails Q. Then P is true in all of the agent's alternatives and since P entails Q, Q is true in all of the agent's alternatives. That is, the agent believes Q. So in the situation theory an agent believes everything that follows logically from his beliefs. If the math professor in our previous example uses the situation theory to reason about his student's beliefs, he will conclude that they believe that every set can be well-ordered as soon as they know the axioms of set theory. There is a similar problem about introspection - as soon as an agent believes P he believes that he believes that he believes..., and so on forever.

This problem is not surprising in a theory that talks about beliefs, but not about the reasoning that creates beliefs. There may well be an infinite set of beliefs that an agent could infer, given arbitrary time and scratch paper. But at any time only a finite number have actually been inferred. If we say nothing about the inference that creates beliefs we can't distinguish between those that are easy to infer and those that take a long time. Then it's no wonder if we end up with a theory saying that everything is inferred in zero time. I conclude that the situation theory of belief is on the wrong track. We need a theory that describes the inference that creates beliefs.

2. The Syntactic Theory

2.1 A Robot and His Beliefs

I have described some of the data that a theory of belief and knowledge must handle, and how Moore fared with the situation theory of belief. Now I consider the syntactic theory. First comes a statement of the theory in English, then the tools needed to formalize it, and then a series of example inferences.

I propose to take very seriously the idea that people are like computers. The agents in my theory look a lot like Von Neumann machines. Not that people are really like Von Neumann machines; rather common sense does not tell us about the massive parallelism and other un-Von Neumann things that go on in our heads. Let us imagine a simple robot, and build a theory that describes his beliefs. We will see that this theory can handle all of the given problems as well as the situation theory, and some of them better.

If we want to write a program that believes that snow is white, we devise a knowledge representation in which we can assert that snow is white - for example, by writing "(white snow)". Then we add this expression to a collection of expressions that are supposed to represent the program's beliefs. This practice suggests a theory: that beliefs are expressions of a knowledge representation language. This is the syntactic theory of belief. It appears now and again in the literature of philosophy - see [7], [3], and [10]. McCarthy [11] was the first AI worker to advocate this theory. Moore and Hendrix [14] argued that the syntactic theory can solve many philosophical problems about belief.

Men, machines and Martians can use very different internal languages to represent the same belief. I propose to ignore this possibility, and assume that all agents use the same representation for every belief. Our robot assumes that everybody else represents beliefs exactly as he does, and he ignores the difference between a belief and his representation of that belief. Konolige [8] was the first to formalize this version of the syntactic theory. His treatment differs from mine in several important ways, which I will note as I come to them.

Suppose John believes that snow is white. The robot thinks that John's representation of this belief is the same as the robot's representation: the expression "(white snow)". The robot also thinks that the representation is the belief. It forms a name for the representation by putting quotation marks around it. So it represents the fact that John believes snow is white by an expression roughly like this:

(believe John "(white snow)")

The first argument of "believe" is the name of a man. The second argument is the name of an expression. To formalize the syntactic theory, one must assign names to expressions. That is, one must devise a system of quotation.

2.2 Formalizing the Syntactic Theory

I use predicate calculus with the following logical symbols:

```
(-> p q) - material implication
(& p q) - conjunction
(V p q) - disjunction
(~ p)    - negation
(all x p) - universal quantification
(some x p) - existential quantification
```

This is the official notation; often I drop parentheses and use connectives as infix operators. A few predicates, like "<" and "=", will also be used as infix operators.

The beliefs of our hypothetical robot are sentences of a first-order logic extended with quotation. These beliefs need not be stored explicitly, but the robot must be able to find out whether he believes a given sentence or not in constant time by a standard retrieval algorithm. We do not say that you believe something if you can infer it after ten minutes of puzzling. All the beliefs are sentences of a single language L. When the robot forms beliefs about its own beliefs, those beliefs must be sentences of L that talk about sentences of L. This is a bit surprising. We are used to talking about an object language L1 by using a meta-language L2, where L1 and L2 are distinct. Why not stick to this method? Since the robot can form beliefs about beliefs about beliefs... up to any finite depth, we could set no limit to the number of meta-languages needed, but that is quite OK. If we follow this plan no language can ever talk about itself, but the robot can always form beliefs about his beliefs by going one step further in the hierarchy. Konolige used such a hierarchy of meta-languages in his formalization of the syntactic theory.

This plan will not work, because it forbids any belief to talk about itself. A belief can talk only about beliefs in languages lower in the hierarchy. In fact beliefs do talk about themselves. For example, a human might notice that he never forgets anything that interests him strongly. Suppose this belief interests him strongly; then it talks about itself, and quite likely makes a true assertion about itself. Or suppose the robot uses a pattern-matcher to retrieve beliefs from memory. It will need a belief describing the pattern-matcher, and this belief can be retrieved by pattern-matching like any other. Thus it says of itself "I can be retrieved by using such-and-such a pattern". There is nothing paradoxical or even unusual going on here. The point is important, because the decision to use a single self-describing language will involve us in the paradoxes of self-reference. One can avoid these paradoxes, but it is not easy.

One way to assign names to sentences is to let sentences be their own names. Then we could represent the fact that John believes snow is white by writing
(believe John (white snow))
This might be a good system, but it is impossible in first-order logic. Sentences denote truth values in first-order logic, they do not denote themselves. We must look farther for a quotation mechanism that will fit into first-order logic.

In English we form the name of a sentence by writing quotation marks around the sentence. Thus the expression
"Snow is white."
denotes the sentence
Snow is white.
If we adopt this scheme in our formal language we could represent the fact that John believes snow is white by writing
(believe John "(white snow)")
We can fit this scheme into first-order logic by saying that

quoted expressions are constants that denote sentences. Yet this idea is not good enough, because it will not allow us to represent the fact that John knows what Mary's phone number is. We observed above that John knows what Mary's number is if he knows that Mary's number is n, where n is an Arabic numeral. We might try to represent this by writing

```
1
(some n (know John "(= (PhoneNumber Mary) n)")
        &
        (IsArabic n)
)
```

But this will not do. By definition of quotation marks, the second argument of the predicate letter "know" denotes the wff
(= (PhoneNumber Mary) n)
This is true no matter what the variable "n" is bound to. So the quantifier "some" does nothing, and (1) means the same as (2).

```
2
  (know John "(= (PhoneNumber Mary) n)")
& (some n (IsArabic n))
```

We need a quotation system that allows us to embed non-quoted expressions in quoted expressions. Then we can represent the fact we tried to represent with (1).

Instead of using a quotation mark that applies to whole expressions, let us quote the individual symbols. If we put the character ' in front of each symbol that we want quoted, we can write

```
3
(some n (know John ('= ('PhoneNumber 'Mary) n))
        &
        (IsArabic n)
)
```

to represent the fact that (1) fails to express. All the symbols in the second argument of "know" are quoted, except for the variable "n" which is bound by the quantifier in the ordinary way. If we can fit this quotation scheme into first-order logic, we can formalize the syntactic theory.

The problem is to assign denotations to the quoted symbols so that sentences like (3) will have the intended meanings, given the usual semantic rules of first-order logic. To each constant of our language we assign a name, formed by appending the character ' to that constant. Thus if "Mary" is a constant and denotes a woman, "'Mary" is a constant and denotes the constant "Mary". To each variable we assign a name in the same way. If "x" is a variable, then "'x" is a constant that denotes the variable "x".

Now consider the symbols that take arguments - function letters, predicate letters, connectives and quantifiers. These symbols are called functors. The term "(& P Q)" consists of the functor "&" and its arguments "P" and "Q". If "F" is a functor of n arguments, then "'F" is a function letter. It denotes the function that maps n expressions el ... en to the expression with functor "F" and arguments el ... en. For example, the function letter "'&" denotes the function that maps wffs wl and w2 to the wff with functor "&" and arguments wl and w2 - which is the conjunction of wl and w2. The function letter "'~" denotes the function that maps a wff to its negation, and so on.

If the variable "n" denotes the arabic numeral "5766", then the term

('= ('PhoneNumber 'Mary) n)

should denote the sentence

(= (PhoneNumber Mary) 5766)

The function letter "'PhoneNumber" denotes the function that maps a term t to the term with function letter "PhoneNumber" and argument t. The constant "'Mary" denotes the constant "Mary". So the term

('PhoneNumber 'Mary)

denotes the term with function letter "PhoneNumber" and argument "Mary", which is

(PhoneNumber Mary)

The function letter "'=" denotes the function that maps terms t1 and t2 to the wff with predicate letter "=" and arguments t1 and t2. So the term

('= ('PhoneNumber 'Mary) n)

denotes the wff with function letter "=" and arguments "(PhoneNumber Mary)" and "5766", which is

(= (PhoneNumber Mary) 5766)

And that is the answer we want.

So if the robot knows what Mary's phone number is, it can represent this fact by the sentence

```
(some n (know Me ('= ('PhoneNumber 'Mary) n))
        &
        (IsArabic n)
)
```

The constant "Me" is the robot's selfname - the robot's usual name for itself. "know" is an ordinary predicate letter - not a special modal operator as in Hintikka. The model theory of our language contains no special rules for interpreting the predicate "know".

On the other hand, suppose that the robot only knows that Mary has a phone number. We represent this as
(know Me ('some 'n ('= ('PhoneNumber 'Mary) 'n)))
In this case the existential quantifier is inside the quotation mark.

The term
4 ('= ('PhoneNumber 'Mary) '5766)
includes the quote name of the arabic numeral for Mary's phone number. The term
5 ('= ('PhoneNumber 'Mary) n)
has a variable in the same position. (4) is the quote name of a wff, but (5) is a wff schema. The quote name of a wff includes a quote name for every term in that wff. A wff schema is like the quote name of a wff, except that variables can appear in place of the quote names of terms. A wff w is called an instance of a wff schema s if for some assignment of values to the free variables in s, s denotes w. For example, if the variable "n" is assigned the value "5766", then (5) denotes
6 (= (PhoneNumber Mary) 5766)
So the sentence (6) is an instance of the wff schema (5).

Writing a quotation mark in front of every functor is a nuisance, so we abbreviate by putting the quotation mark in front of a whole expression. Thus "'(PhoneNumber Mary)" abbreviates "('PhoneNumber 'Mary)". I use infix notation for the connective "&", but never for the quoted function letter "'&". People don't usually use infix notation for function letters, and I want to emphasize that quoted function letters really are function letters. They obey every syntactic and semantic rule that governs function letters in first-order logic. In particular, we apply quotation marks to quoted function letters like any other function letter. Thus "''Superman" denotes the quoted constant "'Superman", which denotes the quoteless constant "Superman", which denotes the man from Krypton.

We also need the function letter "quote", which denotes the function that maps an expression to its quote name. This function maps the wff "(white snow)" to the term "('white 'snow)", for example. So we write
(quote ('white 'snow)) = (''white ''snow)
The argument of "quote" is a term that denotes the wff "(white snow)". The right-hand argument of the equals sign denotes the term "('white 'snow)". This sentence says that the quote name of "(white snow)" is "('white 'snow)" - which is true.

The difference between the quotation mark ' and the function letter "quote" is this. If "v" is a variable, then "'v" is a constant that denotes that variable. "(quote v)" is a term in which the variable "v" is free, and its value depends on the value of "v". If the value of "v" is the constant "Superman", then "(quote v)" will denote the quote name of the constant "Superman", which is "'Superman".

3. Applying the Syntactic Theory

I have now explained the syntactic theory and the machinery used to formalize it. The next task is to apply the formalized theory to the examples described in section 1.2.

3.1 Observation

The robot forms new beliefs by observing the external world and his own internal state. The world is always changing, so the robot needs a theory of time, and it must be able to perceive the passage of time.

3.1.1 Time

Time is a set of instants totally ordered by <. If instant i precedes instant j there is an interval whose lower endpoint is i and whose upper endpoint is j. It contains the instants that are later than i and earlier than j. The lower endpoint of interval I is -I, and its upper endpoint is +I. Nearly all properties of objects hold during intervals. In particular, we write (believe A S I) to indicate that agent A believes sentence S during interval I. Actions happen during intervals. Thus we write (puton Robot A B I) to indicate that the robot puts block A on block B during interval I.

We can define the order relations between intervals in terms of the < relation between their endpoints. For example, interval I is before interval J if the upper endpoint of I is before the lower endpoint of J: +I < -J. Interval I meets interval J if the upper endpoint of I is the lower endpoint of J: +I = -J.

The robot has sensors - devices that detect events in the outside world and produce descriptions of those events in the robot's internal language. The sensors accept physical events as input and produce sentences as output. These sentences become beliefs. A belief created by perception must note the time of the perception. For suppose the robot receives the same message from his sensors at two different times - hears two rifle shots in succession, for example. If the beliefs created by these two perceptions do not mention the times at which the perceptions happened, they will be identical. Then the robot's collection of beliefs will be the same as if it had heard only one shot.

Therefore the robot will need names for intervals of time. These names are constants of the internal language called time stamps. If the robot hears the doorbell ring during interval I, it creates a time stamp for interval I - say "Interval101". Then it adds to its beliefs the sentence

(ringing Me Interval101)

which says that there is a ringing sound in the robot's neighborhood during Interval101. The robot automatically records every perception, and also other events such as inferences and commands to the effectors. Whenever it records such an event it creates a time stamp for the interval when the event happened. It uses that time stamp to name the interval in the belief that records the event.

A time stamp is a useful name for an interval because the robot keeps records of the lengths and order of intervals, and uses time stamps to name the intervals in those records. If the

robot creates a time stamp "Interval53" for an interval J, then as soon as interval J is over the robot forms a belief that records its length. This estimate of the interval's length need not be accurate. People can't tell a minute from fifty seconds without a watch, but they can tell a minute from a second. The robot can get by with rough estimates too. Let us choose a small unit of time and approximate the lengths of intervals with whole numbers of units. Then if J is 30 units long, there is an interval K such that J meets K and
(believe Robot '(= (length Interval53) 30) K)
This belief gives the length of the interval in units, using an arabic numeral to name the number of units. For any integer n, let (arabic n) be the arabic numeral that denotes n. So (arabic 2+2) = (arabic 4) = '4. Suppose the robot creates a time stamp t for an interval i whose length is n units. Then there is an interval j such that i meets j and
(believe Robot ('= ('length t) (arabic n)) j)
Setting t = 'Interval53, n = 30, j = K gives
(believe Robot ('= ('length 'Interval53) (arabic 30)) K)
Since (arabic 30) = '30, we have
(believe Robot ('= ('length 'Interval53) '30) K)
which is a notational variant of the last example.

The robot also records the order relations between intervals that have time stamps. To record the order relation between two intervals it is enough to record the order relations between their endpoints. Given intervals I,J we must record the order relations between -I and -J, -I and +J, +I and -J, +I and +J. Consider the first case. If i and j are intervals with time stamps t1,t2, the robot will record the order relation between +i and +j immediately after the later of the two instants. There are three cases to consider. If +i < +j there is an interval k whose lower endpoint is +j, and

(believe Robot ('< ('+ t1) ('+ t2)) k)
If +i = +j there is an interval k whose lower endpoint is +i, and

(believe Robot ('= ('+ t1) ('+ t2)) k)
Finally, if +j < +i there is an interval k whose lower endpoint is +i, and
(believe Robot ('< ('+ t2) ('+ t1)) k)
So the robot always knows the order relations among all intervals that have been assigned time stamps. Thus the robot has a sense of time: if it remembers two perceptions it remembers which came first and how long they lasted. This particular axiomatization of the sense of time is crude, but it will do for our purposes. One could do a better job with the same formalism if necessary.

3.1.2 Perception

Certain physical events cause the robot's sensors to produce sentences that describe those events. Let us write (perceive Robot s i) to indicate that during interval i the robot's sensors produce the sentence s as a description of some event or state in the outside world. As an example, let us describe the robot's ability to read. The symbols we read and write are expressions of English, not expressions of the robot's internal language. Let us gloss over this distinction and pretend that expressions of the thought language can be written on paper, and the robot can read them.

Suppose that the robot's field of view is a rectangle, and the sensors use integer Cartesian coordinates to describe positions in the field of view. Let (written e x y i) indicate that the expression e is written down at coordinates (x,y) in the

robot's field of view during interval i. If this is the case the robot's sensors will report it, using a quote name for the expression e, arabic numerals for the integers x and y, and a time stamp for the interval i. Suppose that e is an expression, x and y are coordinates, and i is an interval. If (written e x y i), there is a time stamp t for the interval i, and
(perceive Robot ('written (quote e) (arabic x) (arabic y) t) i)
Suppose that "(white snow)" is written at coordinates (150,150) in the robot's field of view during interval I. Then there is a time stamp for interval I, say "Interval99", and we have

```
(perceive Robot
          ('written (quote '(white snow))
                    (arabic 150)
                    (arabic 150)
                    'Interval99
          )
          I
)
```

Using (quote '(white snow)) = ''(white snow) and (arabic 150) = '150 gives

```
(perceive Robot
          ('written ''(white snow) '150 '150 'Interval99)
          I
)
```

The robot believes what its sensors tell it. That is, if it perceives a sentence s during interval i, there is an interval j such that i meets j and the robot believes s during j. In this case there is an interval K such that I meets K and
(believe Robot ('written ''(white snow) '150 '150 'Interval99) K)

3.1.3 Retrieving Beliefs From Memory

The robot acts by executing programs, and its programming language is quite conventional. There is a fixed set of registers. Just like a Von Neumann machine, the robot must bring a data structure into a register before it can operate on that data structure. Remembering a belief means bringing it from memory into a register. If the robot has Bill's phone number stored in its memory, but for some reason can't retrieve it, it cannot call Bill. It has no way to pass the phone number to its telephone dialing routine. This matches our intuitions about people: if you know Bill's phone number, but you can't remember it at the moment, then you can't call Bill.

The statements of the programming language are terms of the internal language, although they have no useful denotations. Considering them to be terms of the internal language is handy because we can then use quotation to name programs. The expressions of the programming language are terms of the internal language, and their values in the programming language are their denotations. Of course they are limited to terms whose values the agent can compute.

All the expressions of the internal language are data structures of the programming language. There are other data structures in the programming language - lists of expressions, for example. Every data structure has a name in the internal language called its print name. The print names of expressions are just their quote names. The print name of the list (cons e nil) is ('cons (PrintName e) 'nil).

The robot uses a statement called the retrieve statement to retrieve beliefs from his memory. A retrieve statement has the form (retrieve r p c), where r is a register, p is a wff schema, and c is a wff. p is called the pattern and c is called the condition. Suppose the robot wants to retrieve a sentence that tells what John's phone number is. Such a sentence has the form

7 ('PhoneNumber 'John n)

The term n must be an arabic numeral:

8 (IsArabic n)

The robot can retrieve a sentence that tells what John's phone number is by executing a retrieve statement with pattern (7) and condition (8):

```
9 (retrieve R1
            ('PhoneNumber 'John n)
            (IsArabic n)
  )
```

A sentence s matches the pattern "('PhoneNumber 'John n)" and the condition "(IsArabic n)" if for some binding of the variable "n", "('PhoneNumber 'John n)" denotes s, and "(IsArabic n)" is true. For example, if "n" is bound to "5766", then "('PhoneNumber 'John n)" denotes "(PhoneNumber John 5766)" and "(IsArabic n)" is true. Therefore "(PhoneNumber John 5766)" matches the pattern "('PhoneNumber 'John n)" and the condition "(IsArabic n)". If a sentence matches the pattern "('PhoneNumber 'John n)" and the condition "(IsArabic n)", then it has the form ('PhoneNumber 'John n) for some arabic numeral n. That is, it tells what John's phone number is. So if the robot knows what John's phone number is, he can retrieve that knowledge by executing the statement (9).

In general, a sentence s matches pattern p and condition c if p is a wff schema and for some bindings of the free variables of p, p denotes s and c is true. Suppose the robot executes the statement ('retrieve r p c) in interval I, and the robot believes a sentence that matches pattern p and condition c. Then the retrieve statement returns a belief that matches the pattern and the condition. There may be several beliefs that match. If so any one of them might be returned. Register r is set to the belief that is returned. That is, there is an interval J such that I meets J and register r holds the returned belief during J. The retrieve statement allows the robot to search his memory.

3.1.4 Introspection

Now that we have a statement that searches the memory we can describe introspection very neatly. All we have to do is say that whenever an agent executes a statement he knows whether it returned a value, and if so what value. The agent can then find out whether he has a certain belief by trying to retrieve it. If he succeeds he will know this, and he can infer that he had the belief; if he fails he will know this also, and he can infer that he had no belief that matched the pattern and the condition.

Suppose, then, that the robot executes a statement s of the programming language during interval I, and it returns a value v. The value v is a data structure. The robot has a time stamp t for the interval I. There is an interval J such that I meets J, and during interval J the robot believes the sentence

('return (SelfName Robot) (quote s) (PrintName v) t)

This sentence says that the robot executed statement s during interval I, and it returned value v. The robot is named by his selfname, the interval by a time stamp, the statement by its quote name, and the returned value by its print name. If the robot executes the statement

```
(retrieve R1 ('PhoneNumber 'John n) (IsArabic n))
```

and it returns the sentence

```
(PhoneNumber John 5766)
```

he will believe

```
(returns Me
        '(retrieve R1 ('PhoneNumber 'John n) (IsArabic n))
        '(PhoneNumber John 5766)
        Interval432
)
```

assuming "Interval432" is the time stamp for interval I. The robot knows that if he executes a retrieve statement during any interval i, and it returns a sentence s, he believed s during i. So he can infer

```
(believe Me '(PhoneNumber John 5766) Interval432)
```

So if the robot believes that John's number is 5766, he can find out that he believes that John's number is 5766.

Suppose the robot executes a statement s of the programming language during interval I,and it returns no value. The robot has a time stamp t for the interval I. There is an interval J such that I meets J and during interval J the robot believes the sentence

```
('~ ('some 'x ('returns (SelfName Me) (quote s) 'x t)))
```

This sentence says that the robot executed statement s during interval I, and it returned no value.

Suppose the robot does not know what John's phone number is. That is, he has no belief of the form ('PhoneNumber 'John n), where n is an arabic numeral. If the robot executes the statement

```
(retrieve R1 ('PhoneNumber 'John n) (IsArabic n))
```

it will return no value. For only a belief of the form ('PhoneNumber 'John n), where n is an arabic numeral, would match the pattern and the condition. If this statement returns no value, the robot will believe the sentence

```
(~ (some x
     (returns Me
              '(retrieve R1 ('PhoneNumber 'John n) (IsArabic n))
              x
              Interval82
     )
))
```

This sentence says that the retrieve statement returned no value. The robot can now argue by contradiction: If I had a belief of the form ('PhoneNumber 'John n), where n was an arabic numeral, it would have matched the pattern "('PhoneNumber 'John n)" and the condition "(IsArabic n)". Then the retrieve statement would have returned a value. But the retrieve statement returned no value. Therefore I have no belief of the form ('PhoneNumber 'John n), where n is an arabic numeral. That is, I do not know John's phone number.

Most theories of belief include an axiom saying that if an agent believes that P, he believes that he believes that P. This theory has instead a general axiom of introspection. It says that if an agent executes a statement of his internal programming language, he knows what value it returned. This axiom allows us to show that if an agent believes that P he can easily discover that he believes that P. We use the same axiom to show that if the agent does not believe that P, he can discover that he does not believe that P. Also we can show that if the agent does not know what X is, he can discover that he does not know what X is - at least in some cases. Later we will find another use for this axiom of introspection.

3.2 Inference

Inference is another process that creates new beliefs. A.I. workers have often distinguished between data-driven and goal-driven inferences. The data-driven inferences happen whenever certain kinds of data are added to the data base. The goal-driven inferences happen when the robot is trying to prove certain kinds of theorems. Data-driven inferences must be limited in some way, because the robot can have only a finite number of beliefs. Breaking up conjunctions is a reasonable data-driven inference: if p & q is added to the belief base, p and q are added too. We can easily describe this with an axiom:

```
(believe Robot ('& p q) i)
-> (believe Robot p i) & (believe Robot q i)
```

The new beliefs formed by breaking up conjunctions could be added explicitly. They could also be added implicitly, by using a belief retrieval program that looks inside conjunctions. Such implementation questions are outside the scope of this theory.

3.2.1 What Do John's Beliefs Entail?

I turn now to the problem of predicting goal-driven inferences. I begin with the usual distinction between search space and search algorithm. To show that an agent will infer a certain belief if he tries to infer it, we must show that there is a path in his search space that leads to that belief, and that his search algorithm is powerful enough to find it. I consider first the problem of showing that the path exists - that is, the agent's beliefs entail the given sentence.

Suppose our knowledge of another agent's beliefs consists of a set of sentences of the form (believe agent (quote s) i) - that is, we have quote names for the sentences the other agent believes. Then by removing the quotation marks we can reconstruct the exact sentences that the other agent believes. We build a data base, separate from our collection of beliefs, containing the sentences that the other agent believes. Any theorem that we can prove using only the sentences in this data base follows from the other agent's beliefs. This is an old idea. Creary [2] was the first to point out that we can combine this kind of reasoning with the use of quotation to represent beliefs.

This method is not sufficient to handle the following inference.

John knows what Mary's phone number is.
John knows that Mary's phone number is the same as Bill's.

John knows what Bill's phone number is.

In our notation the first sentence becomes

```
(know John
      ('= '(PhoneNumber Mary) (arabic (PhoneNumber Mary)))
      I
)
```

We cannot reconstruct the sentence that John believes from this description, because the description doesn't tell us which arabic numeral appears in John's belief. So we can't build a data base containing John's beliefs.

Konolige [8] suggested a solution to this problem. Instead of using simulation, he proposed to describe the proof rules of the language, and use this description to show that a theorem can be proved from another agent's beliefs. For example, we might describe the rule of Modus Ponens by writing

```
(all p q (ModusPonens p ('-> p q) q))
```

This axiom says that p and ('-> p q) entail q by the rule of Modus Ponens. We do not need quote names for the wffs p and q to use this axiom - any names at all will do. If we follow Konolige the lack of quote names for John's beliefs creates no special problem.

The difficulty with Konolige's proposal is that it leads to very long proofs. Suppose we try to find the conclusion of an n-step proof using axioms that describe the proof rules. For each step of the original proof we must build a short proof, which shows that that step produces a certain conclusion. Suppose the average length of these proofs is p steps. Then the proof that our n-step proof has a certain conclusion will involve p X n steps.

Now consider what happens when we nest this kind of reasoning. Suppose John knows that Mary knows his phone number. Then John expects Mary to know how to call him. Suppose we apply Konolige's technique to this problem. There is an n-step proof whose premisses are among Mary's beliefs, and whose conclusion says that Mary can call Bill by dialing a certain number. To show that the proof has this conclusion, John must build a proof of p X n steps. To show that John can build this proof, we must build a proof of p x (p X n) steps. The size of the proofs grows exponentially with the depth of nesting. This is clearly intolerable.

There is an obvious way out of this problem, although it is not trivial to show that it is correct. We pick a new constant, say "C", and use it to stand for the arabic numeral that John uses to name Mary's phone number. Then we can build a data base that approximates John's beliefs. It will contain the sentences

```
(= (PhoneNumber Mary) C)
(= (PhoneNumber Mary) (PhoneNumber Bill))
```

from which we can infer

```
(= (PhoneNumber Bill) C)
```

Since "C" stands for an arabic numeral, John can infer a sentence of the form ('= ('PhoneNumber 'Bill) n), where n is an arabic numeral. That is, John can figure out what Bill's phone number is.

Let us state the argument more precisely. If we were to go through the proof we have just built, and replace the constant "C" with the arabic numeral that appears in John's belief about Mary's phone number, the result would be a new proof. John believes the premisses of this proof, and its conclusion gives an arabic numeral for Bill's phone number. So there is a way to prove from John's beliefs a theorem that gives an arabic numeral for Bill's phone number. The crucial assumption here is that if we go through the proof and replace "C" with another constant, the result is still a proof. In the appendix I show that if we take any proof and replace all occurrences of a constant with a closed term, the result is still a proof. This theorem justifies the use of a new constant to represent an unknown term that appears in another agent's beliefs. It allows us to prove the correctness of an axiom schema called the Reflection Schema, which does the kind of reasoning that we have informally described.

3.2.2 The Reflection Schema

Consider first the simple case of the Reflection Schema, in which we have the quote names of the other agent's beliefs. The proof to be reflected consists of a single step. The rule of Substitution Of Equals is applied to the premisses

```
(= (PhoneNumber Mary) 5766)
(= (PhoneNumber Bill) (PhoneNumber Mary))
```

producing the conclusion

```
(= (PhoneNumber Bill) 5766)
```

The structure of proofs is described exactly in the appendix. For now it is enough to say that a proof is formed by starting with wffs called premisses and repeatedly applying proof rules. Every proof has a print name, which includes the quote names of all the premisses of the proof. Given the print name of a proof one can easily reconstruct that proof, just as one can reconstruct a sentence from its quote name. The print name of the proof just given is

```
(EqSubst '(= (PhoneNumber Mary) 5766)
         '(= (PhoneNumber Bill) (PhoneNumber Mary))
         '(= (PhoneNumber Bill) 5766)
)
```

(IsProof p) means that p is a correct proof. If p is a correct proof, the sentence

('IsProof (PrintName p))

is an instance of the Reflection Schema. For the proof given above we have the instance

```
10 (IsProof (EqSubst '(= (PhoneNumber Mary) 5766)
                     '(= (PhoneNumber Bill) (PhoneNumber Mary))
                     '(= (PhoneNumber Bill) 5766)
            )
   )
```

If John believes the premisses of this proof he can infer its conclusion - that is, he can infer that Bill's number is 5766.

It is easy to implement this schema. The implementation is a program that takes a sentence as input and decides whether it is an instance of the Reflection Schema. The input sentence contains the print names of a proof and its conclusion. From the print names the program reconstructs the proof and the conclusion. Then it calls the programs that implement the other proof rules to decide whether the proof is correct. If it is, the input sentence is an instance of the Reflection Schema.

Suppose John knows Mary's phone number. Then he believes the sentence

```
('= '(PhoneNumber Mary) (arabic (PhoneNumber Mary)))
```

If he also believes that Bill's phone number is the same as Mary's, he believes the sentence

```
'(= (PhoneNumber Bill) (PhoneNumber Mary))
```

By Substitution Of Equals he can infer

```
('= '(PhoneNumber Bill) (arabic (PhoneNumber Mary))
```

The version of the Reflection Schema that we have just seen is not strong enough to prove this. It demands the quote names of the sentences in the proof to be reflected, and we do not have the quote name of the arabic numeral for Mary's phone number. We need a stronger Reflection Schema, which includes the following instance.

```
11
(all x
  (ClosedTerm x)
  -> (IsProof
       (EqSubst ('= '(PhoneNumber Mary) x)
                ('= '(PhoneNumber Bill) '(PhoneNumber Mary))
                ('= '(PhoneNumber Bill) x)
       )
     )
)
```

Since an arabic numeral is a closed term, we infer

```
(IsProof
  (EqSubst ('= '(PhoneNumber Mary) (arabic (PhoneNumber Mary)))
           ('= '(PhoneNumber Bill) '(PhoneNumber Mary))
           ('= '(PhoneNumber Bill) (arabic (PhoneNumber Mary)))
  )
)
```

And this is the desired conclusion.

The argument of the predicate letter "IsProof" in (11) is called a proof schema. A proof schema is like the print name of a proof, except that a variable can appear instead of the quote name of a term. That is, a proof schema is to the print name of a proof what a wff schema is to the quote name of a wff. The argument of "IsProof" in (10) is the print name of a proof. It gives the quote name "'5766" of the arabic numeral for Mary's phone number. The argument of "IsProof" in (11) is a proof schema. The variable "x" appears in place of the quote name "'5766".

Implementing this version of the Reflection Schema is not as easy as implementing the first version. The first version reconstructed the proof to be reflected from its print name, and then called the other proof rules to decide whether the proof was correct. The new version gets only a proof schema, which stands for a whole class of proofs called instances of the schema. A proof p is an instance of a proof schema s if for some bindings of the free variables of s, s denotes p. The proof named in (10) is an instance of the proof schema that appears in (11). It is formed by binding the variable "x" to the term "5766". An instance of the Reflection Schema is true only if all instances of its proof schema are correct proofs.

The solution is to form a proof called the typical instance of the proof schema. Every instance of the schema can be formed by substituting closed terms for constants in the typical instance. If the typical instance is a correct proof, then since substitution maps proofs to proofs, all the instances are correct proofs.

The typical instance of a proof schema is its denotation in an environment that binds its free variables to new constants. For example, if we bind the variable "x" to the new constant "C", then the proof schema in (11) denotes a proof whose premisses are

```
(= (PhoneNumber Mary) C)
(= (PhoneNumber Mary) (PhoneNumber Bill))
```

Its conclusion is

```
(= (PhoneNumber Bill C)
```

and the rule used is Substitution Of Equals. Since this proof is correct, all instances of the proof schema in (11) are correct. Therefore (11) is a true sentence (in the intended model). This is a rough explanation of the Reflection Schema, omitting many complications. The full story, and a proof of correctness, appear in the appendix.

We have shown how to make inferences by simulation from atomic sentences like

```
(believe John
         ('= '(PhoneNumber Mary) (arabic (PhoneNumber Mary)))
         I
)
```

and from conjunctions of such sentences. What about the other connectives and quantifiers? We ought to be able to make inferences from disjunctions and negations of sentences about belief, and from universally or existentially quantified statements about belief. Here we see the value of adding the Reflection Schema to a first-order logic. We can handle negation, disjunction and quantification without adding any more rules or axioms.

Consider negation. If John believes that Mary's number is 5766, and he does not believe that Bill's number is 5766, he must not believe that Bill's number is Mary's number (assuming that he can make trivial inferences). First-order logic allows us to argue by contradiction: to prove ~p by assuming p and proving a contradiction. So assume

```
(believe John '(= (PhoneNumber Mary) (PhoneNumber Bill)) I)
```

and

```
(believe John ('= ('PhoneNumber 'Mary) '5766) I)
```

We can prove by simulation that there is a one-step argument from these beliefs of John's to the conclusion

```
(= (PhoneNumber Bill) 5766)
```

So if John can find this one-step argument, he believes that Bill's number is 5766. This contradicts the assumption that he has no such belief, so we can conclude that John does not believe that Bill's number is Mary's number.

Suppose John believes that all millionaires are happy, and he either believes that Bill is a millionaire or that Bob is a millionaire - we don't know which.

```
  (believe John ('all 'x ('-> ('millionaire 'x) ('happy 'x)) I)

  (believe John ('millionaire 'Bill) I)
V (believe John ('millionaire 'Bob) I)
```

We should be able to prove that he believes someone is happy (again assuming he can make trivial inferences).

```
(believe John ('some 'x ('happy 'x)) I)
```

First-order logic allows us to argue by cases - to prove p from q V r by proving p from q and proving p from r. Assume first that

John believes that Bill is a millionaire. Then we can build a data base that represents his beliefs, and it looks like this:

```
  (all x (millionaire x) -> (happy x))
  (millionaire Bill)
```

In this data base we can easily infer that someone is happy:
(some x (happy x))
So John believes that someone is happy. Now suppose John believes that Bob is a millionaire. We can prove again that John believes someone is happy. Since John either believes that Bill is a millionaire or that Bob is a millionaire, it follows that he believes someone is happy.

Suppose John knows every millionaire by name. That is, for each millionaire he believes that N is a millionaire, where N is the millionaire's personal name. Let (name x) be x's personal name, for any person x. (name John Smith) is "John Smith", and so on. Then we can describe John's exhaustive knowledge of millionaires by writing:

```
(all x (millionaire x)
       -> (believe John ('millionaire (name x)) I))
```

John also believes that every millionaire is happy.
(believe John '(all x (-> (millionaire x) (happy x))) I)
We should be able to infer that for each millionaire, John believes that he is happy.
(all x (millionaire x) -> (believe John ('happy (name x)) I))
First-order logic allows us to prove that everything has a certain property by proving that an arbitrary object has that property. In the first-order proof system used here, free variables represent arbitrary objects. So let z be an arbitrary object, and suppose z is a millionaire. Since John knows every millionaire by name, we have
(believe John ('millionaire (name z)) I)

We choose the constant "C" to stand for the unknown term (name z). Then the data base that represents John's beliefs contains the sentences

```
(millionaire C)
(all x (millionaire x) -> (happy x))
```

These sentences entail

```
(happy C)
```

We conclude

```
(believe John ('happy (name z)) I)
```

But since z represents an arbitrary millionaire, we have

```
(all x (millionaire x) -> (believe John ('happy (name x)) I))
```

which was to be proved. The treatment of existentially quantified sentences about belief is similar.

By adding the rule of Reflection to a first-order logic, and proving that it is correct, we gain two advantages. Because we have proved the rule correct, we can rule out the possibility of bugs caused by bad interactions between Reflection and some obscure feature of the logic. Because the rule is part of a system that represents negation, disjunction and quantification correctly and completely, no extra work is needed to handle negations, disjunctions and quantifications of statements about belief. Imagine what would have happened if we had first written a rough description of our inference rules in English, then written 30 pages of LISP to implement them, and then started "maintaining" the code so that it changed once a week. How would one show that replacing constants with closed terms maps proofs to proofs? The kind of work we have just done is possible only if the rules of inference are set down plainly. These considerations prove nothing about the merits of frames vs. semantic nets vs. logic. They do indicate that an if an AI system is going to use reflection to reason about belief, its inference rules must be made explicit, not hidden in the code.

3.2.3 What Can John Infer from His Beliefs?

The rule of Reflection allows us to show that an agent's beliefs entail a sentence. To show that an agent will actually infer that sentence, we need to show also that his theorem prover is powerful enough to find a proof. The agent calls his theorem prover by executing a prove statement. This statement has the form ('prove r p c), where r is a register, p is a pattern, and c is a condition. When the agent executes this statement his theorem prover tries to prove a sentence that matches the pattern and the condition. If it succeeds, the prove statement returns that sentence and puts it in register r. Also the sentence is added to the agent's beliefs.

This leaves the crucial questions: can the theorem prover prove a sentence that matches the pattern and the condition? If it can, which one will it prove? Creary [2] offered a simple answer. If agent A can prove by simulation that agent B's beliefs entail P, then agent B can prove P. This is very different from saying that if agent B's beliefs entail P, agent B can prove P. It is quite possible that B's beliefs entail P but A cannot prove this fact.

Agent A predicts the behavior of agent B's theorem prover by making an empirical observation of the behavior of his own theorem prover. This involves the assumption that agent A is not much brighter than agent B - an assumption that is reasonable in most common sense contexts, though not when A is a math teacher and B is a student. Agent A can answer the question "Which theorem will agent B prove?" by similar reasoning. Perhaps there are many theorems entailed by agent B's beliefs that would match the pattern and condition that B gave to his theorem prover. If A has shown that a particular theorem entailed by B's beliefs will match the pattern and condition, he can assume that this theorem is an obvious answer, and it is the one B will prove. This is not so convincing as the last assumption, but it is the same principle - A is predicting the behavior of B's theorem prover by observing the behavior of his own theorem prover. Agent A can even make a rough estimate of the time it will take for B to prove the theorem: it should be no more than the time it took A to simulate B's reasoning. We already have an axiom of introspection, which says that when agent A executes a statement of his programming language he knows what it value it returned. So if agent A executes the prove statement he knows what theorem he proved. Also he has a time stamp for the interval in which he executed the prove statement, so he knows how long it took.

Suppose then that A and B are two agents, and the following conditions hold.

i. Agent B executes the statement ('prove r p c) during an interval i, and sentence s matches the pattern p and condition c.
ii. Agent B's beliefs entail sentence s.
iii. Agent A has proved during an interval j that agent B's beliefs entail s.

Then agent B's execution of ('prove r p c) returns the value s, and interval i is no longer than interval j. The first condition says that agent B is trying to prove sentence s, or one like it. The second condition says that there is a proof of s from agent B's beliefs. The third condition says that agent A has found such a proof, so it is not too difficult.

Now we can do the following example from part 1.

```
John knows that Mary's number is 5766.
John knows that Mary's number is the same as Bill's number.
John is trying to figure out what what Bill's number is.
-------------------------------------------------------------
John will infer that Bill's number is 5766.
```

We represent the statement that John is trying to figure out what P is by saying that John has called his theorem prover and asked it to prove a sentence that tells what P is. In this case, John wants his theorem prover to prove a sentence that tells what Bill's phone number is. A sentence that tells what Bill's phone number is must have the form

```
('= '(PhoneNumber Bill) n)
```

where n is an arabic numeral. John can express this requirement by giving his theorem prover the pattern

```
('= '(PhoneNumber Bill) n)
```

and the condition

```
(IsArabic n)
```

So we formalize the statement that John is trying to figure out what Bill's number is by writing

```
(execute John
         '(prove R1 ('= '(PhoneNumber Bill) n) (IsArabic n))
         I
)
```

This satisfies the first condition.

As usual, we formalize the first two premisses as

```
(believe John '(= (PhoneNumber Mary) 5766))
(believe John '(= (PhoneNumber Mary) (PhoneNumber Bill))
```

By the Reflection Schema the robot can prove

```
(proof (EqSubst '(= (PhoneNumber Mary) 5766)
                '(= (PhoneNumber Mary) (PhoneNumber Bill))
                '(= (PhoneNumber Bill) 5766)
       )
)
```

Since agent B believes the premisses of this proof, his beliefs entail its conclusion "(= (PhoneNumber Bill) 5766)". This satisfies the second condition. After the robot's theorem prover proves this theorem, he will know that it was proved. By the axiom of introspection he will believe the sentence

```
(returns Me
         '(prove ....)
         '(proof (EqSubst ...))
          Interval236
)
```

This sentence says that the robot has proved that John's beliefs entail the sentence (= (PhoneNumber Bill) 5766). It satisfies the third condition. Now the robot can infer that John's theorem prover will return the sentence (= (PhoneNumber Bill) 5766), and as a result John will believe this sentence.

The last argument of the predicate letter "returns" is a time stamp for the interval when the robot executed the prove statement. The robot's sense of time tells him how long this interval was. So he believes, let us say, the sentence
(length Interval236) = 20
Then he can infer that John will take no more than 20 units of time to figure out what Bill's phone number is.

An agent figures out what inferences another agent can make by simulating his reasoning. If the other agent's beliefs include terms that are unknown to the simulator, he must use an approximation of the other agent's beliefs in his simulation. The simulator introduces new constants to represent the unknown terms in the other agent's beliefs. By noting the time that it took him to simulate the other agent's reasoning, the simulator judges how hard it will be for the other agent to find the same line of reasoning. The simulator does not have or need a theory that explains why one line of reasoning is harder to find than another. He uses empirical observations of the behavior of his own theorem prover to predict the behavior of another agent's theorem prover. So we have a theory of belief that talks about the processes that create beliefs.

3.3 Knowing What

John knows what Mary's phone number is if he knows that Mary's number is n, where n is an arabic numeral. We represent this as

```
(some n   (know John ('= '(PhoneNumber Mary) n))
        & (IsArabic n)
)
```

We can give a similar treatment of other "knowing what" examples. An English teacher would say that a student knows who the author of "Hamlet" is if he knows that the author of "Hamlet" is n, where n is a personal name. We can represent this as

```
(some n   (know student ('= '(author Hamlet) n)
        & (IsPersonalName n)
)
```

Since "Shakespeare" is a personal name, the student knows who the author of "Hamlet" is if he knows that the author of "Hamlet" is Shakespeare. When we say that someone knows what X is, we mean that he knows that X is n, where n is a name or description having some property P. In the first case, P is the property of being an arabic numeral. In the second case, P is the property of being a personal name. Kaplan [7] suggested this approach.

As we saw in the example of being lost in the city, the property P depends on context. In that example, the agent wanted to use the name n to accomplish a task. First the task was to get back to the hotel, and he wanted n to be something like "five blocks north of the hotel on High Street". Then he switched to a new task: helping Mary to find John. For this task the name "here" described John's location quite well. This example suggests that John knows what X is if he knows that X is n, where the name or description n contains the information needed for the task at hand.

That would certainly explain why knowing an arabic numeral for Mary's phone number counts as knowing what her number is. The arabic numeral allows us to call Mary, which is what phone numbers are for. Or suppose John tells me that he lives in the grey house across the street from the Star Market on Park Avenue. Then if I know how to get to the Star Market I can get to John's house. I could also claim that I know where John lives, even if I have never seen his house. This example fits the proposal nicely.

According to Konolige I know where John lives only if I have a standard name for John's house - one that denotes the same house in all possible worlds. Certainly "the grey house across from the Star Market" does not denote the same house in all possible worlds, so Konolige predicts that in this case I do not know where John lives. Since Konolige does not suggest that the set of possible worlds under consideration can vary with context, he does not allow context to determine whether knowing that X is n entails knowing what X is. Moore's proposal is that I know what X is if X is the same object in all my alternatives. This is different from Konolige's idea, because my alternatives are a small subset of the set of all situations. It still does not explain how I can know where John lives when I have never seen his house, and can only describe it as "the grey one across from the Star Market".

Alas, there are plenty of examples where my proposal fails. Often there is no particular task at hand. In a discussion of politics I may ask "Do you know who the Saudi oil minister is?" Presumably I want his personal name, but there is no obvious task to be accomplished with this information. At least my proposal accounts for the importance of context in deciding what knowledge one needs about an object in order to know what that object is. The proposal is not helpful unless, having identified the task at hand, we can decide what knowledge is needed to do that task. This is the subject of the next section.

One can give a similar account of de re belief reports. If you think, in the de re sense, that John's sister is his wife, you have a belief of the form "n is John's wife", where the name n really denotes John's sister. Let "denotation" name the function that maps a term to its denotation. This function maps the name "Superman" to the man from Krypton, for example. We represent the fact that I think John's sister is his wife by

```
(some n   (believe I ('= n '(wife John)))
        & (denotation n) = (sister John)
)
```

There must be some limitations on the choice of the name n. For example, if Bill is in fact the president of IBM, the name "president of IBM" denotes him. Still believing the tautology

"The president of IBM is the President of IBM" will not qualify you as believing that Bill is the president of IBM. The conditions on the name n seem to be weaker in this case than in the "knowing what" examples.

3.4 Knowing How

If you know that Mary's number is 5766, you know how to call Mary. When does knowing that P entail knowing how to perform action A? Moore proposed that actions have parameters - for example, the number to be dialed is a parameter of the action of dialing. You know how to do the action if you know what the parameters are. And you know what X is if X denotes one object in all your alternatives. Since "5766" denotes one object in all situations, someone who knows that Mary's number is 5766 knows how to call Mary. Konolige agrees that you know how to perform an action if you know what its parameters are. As mentioned above, Konolige holds that you know what X is if you have a standard name for X. Since "5766" is a standard name, someone who knows that Mary's number is 5766 knows how to call Mary.

Both proposals go wrong in the same way. "Six times thirty-one squared" denotes 5766 in all situations. So if I know that Mary's number is six times thirty-one squared I know how to call Mary, according to both Moore and Konolige. And this prediction is wrong. I have to figure out that six times thirty-one squared is 5766 before I can call Mary, and if I don't have pencil and paper handy it may not be easy to call her.

Even if these proposals could be made to work, they are not satisfying. There is no apparent reason why having a standard name should help you to perform an action. A better theory would have more intuitive appeal. It would make us say "Ah, now I see why you need that piece of knowledge to perform that action."

Let us return to our imaginary robot, and ask "How would the robot call Mary on the phone? At which point would he use his knowledge of her phone number?" The robot can act only by executing a program. He knows how to perform an action if he knows what program he should execute to perform that action. Since programs are expressions of the internal language, there is no mystery about when the robot knows what program to execute. He knows what program to execute if he has the quote name of the program. From the quote name he can reconstruct the program itself; he can then proceed to execute it. Our problem is then to show that the robot can construct a program for dialing Mary's number if he knows the arabic numeral for Mary's number.

Intuitively it is obvious why you need to know the arabic numeral for Mary's number to call her. Telephones have arabic numerals printed on their dials. You use those numerals to identify the right holes to put your finger in. If phones had roman numerals printed on them instead, you would need the roman numerals for Mary's number to call her. We must reconcile this common sense observation with the claim that the robot knows how to dial Mary's number if he knows what program to execute in order to dial Mary's number.

The robot performs physical actions by sending commands to his effectors. These are devices that accept commands in the internal language as input, and produce physical actions as output. A command is simply a sentence of the internal language that describes the desired action. If the effectors perform this action the sentence will be true.

Of course the effectors can only accept certain sentences as commands. Even if two sentences describe the same action, it does not follow that if the effectors can accept one sentence as a command they can also accept the other. A real robot can turn one joint of his arm through an angle of n degrees by putting a binary numeral for n in a certain register. No other name for the number n will do.

Actually the commands are not sentences but wffs. A sentence that describes an action must give the time when the action was performed. But when the robot sends a command to his effectors he wants it carried out now; he does not need or want to specify the time. So the robot uses the free variable "Vnow" to stand for the present time in commands to the effectors. If the robot sends a command to his effectors during interval I, the action will be carried out during interval I. So the command will be satisfied when the variable "Vnow" is bound to the interval I. "(CloseHand Robot I)" means that the robot closes his hand during interval I. The robot uses his selfname to refer to himself in commands to his effectors. So he sends the wff "(CloseHand Me Vnow)" to his effectors when he wants to close his hand. "(command Robot w i)" means that the robot sends wff w to his effectors during interval i. So the robot believes

(all i (command Me '(CloseHand Me Vnow) i) -> (CloseHand Me i))

This sentence says that if the robot commands his hand to close, it will close.

Let us return to the phone dialing problem. Suppose for simplicity that the phone has push buttons rather than a dial. To "dial" Mary's number the robot must tell his hand which buttons to push. The robot's hand is presumably guided by his eye. We assume that the problem of hand-eye coordination is handled by low-level routines that do not concern us. All the robot has to do to direct his hand to a certain object is to supply the coordinates of that object in the visual field. As mentioned above, the visual field is a rectangle, and the robot uses Cartesian coordinates to specify positions in the visual field. "(push Robot x y I)" means that during interval i the robot's hand reaches out and pushes the object at coordinates (x,y) in the robot's field of view. When the robot commands his hand to push the object at coordinates (x,y), he uses arabic numerals to specify the coordinates x and y. So if the robot issues the command

```
('push 'Me (arabic x) (arabic y) 'Vnow)
```

his hand will push the object at coordinates (x,y).

When the robot points his eye at the buttons on the phone, he will see the arabic numerals from "0" to "9" printed on the buttons. As stated in section 3.1.2, the sensors will report that a numeral n is written at coordinates (x,y) by producing the sentence

```
('WrittenAt (quote n) (arabic x) (arabic y) 'Interval99)
```

The numerals (arabic x) and (arabic y) are precisely the data structures the robot needs to build a command that will cause his hand to push the button with the numeral n printed on it. If the sensors produce the sentence

```
(WrittenAt '5 128 100 Interval99)
```

the robot knows that the button with the numeral "5" printed on it is at coordinates (128,100) in his field of view. He can push it by issuing the command

```
(push Me 128 100 Vnow)
```

So assume that the robot is looking at the telephone. He can dial the number "5766" by executing a program that looks roughly like this:

```
Scan until you receive a percept of the form
(WrittenAt '5 x1 y1 i);
Send the command (push Me x1 y1 Vnow);
Scan until you receive a percept of the form
(WrittenAt '7 x2 y2 i);
Send the command (push Me x2 y2 Vnow);
Scan until you receive a percept of the form
(WrittenAt '6 x3 y3 i);
Send the command (push Me x3 y3 Vnow);
Scan until you receive a percept of the form
(WrittenAt '6 x4 y4 i);
Send the command (push Me x4 y4 Vnow);
```

Now it is clear why the robot needs the arabic numerals for the digits of Mary's phone number to construct a program for dialing her number. He needs to find the the right buttons to push, and he identifies them by the arabic numerals printed on them. The robot cannot dial the number in a pitch black room. Moore and Konolige treat dialing as a primitive action. They do not mention that you *have to look for* the buttons with the right numbers printed on them, so they do not predict any difficulty about dialing a telephone in the dark. They can of course assert that having light is a precondition of dialing. But it is better to derive this precondition from the general rule that you can't see in the dark.

3.5 Belief and Truth

Common sense says that snow is white if and only if it is true that snow is white. One can formalize this idea with a Truth Schema. For every sentence p, the sentence

```
('<-> ('true (quote p)) p)
```

is an instance of the Truth Schema. This schema says that the truth of a sentence depends on the properties of the objects mentioned in the sentence. For example, one instance of the Truth Schema is the sentence

(true ('white 'snow)) <-> (white snow)

which says that the sentence "(white snow)" is true iff snow is white.

Now we can formalize the inferences involving truth in section 1.2. The following inference is correct:

```
John believes that gold is an element.
Everything that John believes is true.
----------------------------------------
Gold is an element.
```

The formal translations of the premisses are:

(believe John ('element 'gold) I)
(all x (believe John x I) -> (true x))

These sentences entail

(true ('element 'gold))

The following is an instance of the truth schema:

(true ('element 'gold)) <-> (element gold)

The last two sentences entail

(element gold)

that is, gold is an element.

The Truth Schema captures our intuitions about truth nicely. Unfortunately, our intuitions about truth are not consistent. The problem is the celebrated liar paradox. Suppose I say "This statement is false". If the statement is true, it is false; and if it is false, it is true. We can get a formal version of this contradiction by assuming

p = '(~ (true p))

The following is an instance of the Truth Schema:

(true '(~ (true p))) <-> (~ (true p))

Substituting equals gives

(true p) <-> (~ (true p))
which is an obvious contradiction.

How we deal with this problem depends on what we think the goal of Artificial Intelligence is. If we are trying to make machines as intelligent as possible, we must abandon the Truth Schema and look for a new schema that can handle the Liar sentence without contradiction. There are several ways to do this. For example, see [15].

On the other hand, if we are trying to make machines as intelligent as people, we don't want to give them a solution of the Liar Paradox even if we know of one. Ordinary people can't resolve the Liar Paradox; they can only note that it is a paradox, and go on using the Truth Schema as before. If our machines are only supposed to be as intelligent as ordinary people, they should do the same. This does not mean that we should put the Truth Schema into our logic and forget about the matter. If we do that, we have no way of knowing when the contradictions will appear or how much trouble they will cause. Even if we find by experiment that no problem arises in this or that application, we can't just ignore the problem. It is our job, not just to build programs that work, but to understand why they work. Our task is not done until we answer the question "How can machines (or people) get away with using an inconsistent theory of truth?".

Let us look again at the Liar example, p = '(~ (true p)). Suppose we try to discover whether this sentence is true by using the usual Tarskian rules for assigning truth values, along with the rule that (true x) is assigned the same truth value as x. The sentence is the negation of '(true p), so to find its truth value we must find the truth value of '(true p). To find the truth value of this sentence we must find the truth value of p. But p is the sentence we started with. The attempt to find the truth value of p thus leads to an infinite recursion. A sentence is called grounded if we can find its truth value by the given rules without infinite recursion.

Kripke [9] pointed out that many quite ordinary utterances can be ungrounded if circumstances are very unfavorable. Suppose Joe Smith is walking down a road at noon on July 1, 1982. He sees a sign by the road, too far away to read, and remarks "The statement on that sign is true." He approaches the sign and reads the words "The utterance of Joe Smith at noon on July 1, 1982 is false". If we attempt to find the truth value of this sentence by usual rules we get an infinite recursion. But of course the example was created only by assuming a very peculiar road sign. Although many utterances could lead to this kind of infinite recursion, in practice not many do.

In the appendix we will offer a formal definition of this notion of a grounded sentence, and prove that in any model we can choose the extension of the predicate "true" so that (true 'p) <-> p holds for every grounded sentence. Since ungrounded sentences seldom arise in practice, they are rare in the intended model of the robot's beliefs. Therefore most instances of the Truth Schema are true in the intended model. And that is why it is safe for the robot to use the Truth Schema.

4. Conclusions and Further Work

This work makes three main improvements in the match between theory and common sense. First, it does not predict that agents instantly believe everything that can be proved from their beliefs. It considers the agent's goals and his limited inference ability before predicting that he will make an inference, and it says that inference takes time. Second, it gives a better account of what you must know about an object in order to know what that object is. It says that you know what an object is if you know enough about it to carry out your intended actions. This is far from complete, but still a real improvement. Finally, it gives a better account of when you need knowledge to perform an action. It simply formalizes the obvious: robots perform actions by sending commands to effectors, and to act they must find out which commands will produce the desired actions.

These improvements have practical importance. A planner will not get far if (following the situation theory) it thinks that there is no point in planning to do inferences, since they all happen instantly and automatically. Nor will an interactive program do well if it thinks that a large mathematical expression is a good answer to a user's question because it is a standard name.

These improvements are all made in the same way: by forgetting about alternative situations and going back to familiar ideas from computer science. If an agent uses sentences to represent his beliefs, and applies inference rules to them, there is no reason to expect that he will believe all consequences of his beliefs. If an agent acts by sending commands to his effectors, then of course he must find out which commands will produce the desired actions. Konolige took this line, but he only went halfway - he returned to the situation theory in his treatment of knowing what and knowing how. As a result, the problems of the situation theory reappear in Konolige's theory.

There is also an important gain in the technique for reasoning about another agent's inferences. The idea of building a data base to represent another agent's beliefs has always appealed to AI workers. But it was not very useful with no way to represent "John knows what Mary's phone number is" in the data base. The use of new constants to stand for unknown terms solves this problem.

Further work on these lines could be of two kinds: improvements in the theory, and applications of the theory. The theory has a major shortcoming as it stands: unlike Moore's theory, it does not include a formal theory of planning. Since my treatment of time uses intervals, not situations, one cannot simply add situation calculus. It would be straightforward to get rid of the intervals and add situation calculus to the theory. But situation calculus has its own problems, and people are working on better treatments of time [16]). It would be nice to keep the interval theory of time and find a planning theory based on intervals rather than situations. This problem is tackled in [12], [1] and [5].

One could make several other extensions to the theory, but real progress will come only from studying applications. A program can use this theory in two ways: to reason about its own beliefs and other people's. Planning programs need to reason about their own beliefs so that they can plan to acquire knowledge, either for its own sake or as a prerequisite to further actions. Since most planning work to date has used situation calculus, one might replace intervals with situations before applying the theory to planning. The theory would then allow a program to build plans involving perception, introspection, inference and physical actions.

Reasoning about other people's beliefs is important in interactive programs (whether or not they use natural language) and in story understanding. With a good representation of belief one can assert (for example) that agent A is lying to agent B, while B realizes that A is lying but pretends to be fooled. Here one would like to use knowledge of an agent's beliefs to predict his actions, a topic considered in [4]. This type of application should provide evidence for a better theory of knowing what.

Some people hope that AI programs in all domains can benefit from knowledge about their own knowledge. One might express heuristics by saying "This piece of knowledge is good for solving this type of problem". One could describe a default by saying "Assume that a human being has two arms unless you have knowledge to the contrary", thus avoiding the pitfalls of non-monotonic logic. These ideas are attractive but untested.

5. Appendix: Proofs

5.1 The Reflection Schema is Correct

Before we can prove that the Reflection Schema is correct, we must be more explicit about the structure of proofs and their print names. A proof consists of a proof rule, a conclusion, and zero or more subproofs. A proof rule that requires no subproofs is an axiom schema.

The print names of proofs are formed as follows. Consider a proof with rule R, subproofs P1 .. Pn, and conclusion q. There is a function letter R' called the proof-building function letter for R. R' denotes the function that takes proofs P1 .. Pn and a sentence q, and returns the proof with rule R, sub-proofs P1 ... Pn, and conclusion q. Then the term
(R' (PrintName P1) ... (PrintName Pn) (quote q))
is the print name of the proof with rule R, sub-proofs P1 ... Pn, and conclusion q.

For example, consider a proof whose rule is Excluded Middle and whose conclusion is (p V ~p). It has no subproofs, since Excluded Middle requires no premisses to derive its conclusion. Its print name is
(ExMiddle '(p V ~p))
"ExMiddle" is the proof-building function letter for the rule Excluded Middle.

In section 3.2.2 I outlined the proof that the Reflection Schema is correct. The core of the argument was that substitution maps proofs to proofs. Our proofs use the rules of first-order logic plus the Truth Schema and the Reflection Schema. So we should begin by picking a first-order proof system and showing that substitution maps proofs to proofs in that system. There is a minor difficulty here. Every first-order proof system has a rule that allows us to prove (all x (p x)) by proving (p t) without using any assumption that contains the term t. In some systems the term t is a constant, in others a variable. We must choose a system in which t is always a variable. For if t is a constant, a substitution might introduce that constant into one of the premisses used to prove (p t). In that case the conclusion (all x (p x)) would not be allowed, and it would not be true that substitution always maps proofs to proofs. So we must use a proof system in which variables, not constants, are used to stand for arbitrary objects. Given such a system, one can show by a straightforward induction that substitution maps proofs to proofs. I omit this argument.

Next consider proofs that use first-order rules and the Truth Schema, but not Reflection. To show that substitution maps proofs to proofs, we need only show that substitution maps instances of the Truth Schema to instances of the Truth Schema. If we allow arbitrary substitutions of closed terms for constants, this is false. Consider a typical instance of the Truth Schema:

(true ('white 'snow)) <-> (white snow)

Suppose we replace the constant "snow" with the constant "coal". We get

(true ('white 'snow)) <-> (white coal)

This is certainly not an instance of the Truth Schema. We should have replaced the constant "'snow" with "'coal" as well. Then we would have gotten

(true ('white 'coal)) <-> (white coal)

which is an instance of the Truth Schema.

To accommodate the Truth Schema, we must substitute (quote c) for (quote d) whenever we substitute c for d. A substitution that obeys this rule is called a Q-substitution. Let (quote^n e) be the result of quoting expression e n times. So (quote^2 "C") is "''C", for example. If s is a substitution of closed terms for constants, let (s e) be the result of applying s to expression e. Substitution s is a Q-substitution if for all expressions e and all integers n

12 (s (quote^n e)) = (quote^n (s e))

A Q-substitution s is completely specified by the values of (s c) for quoteless constants c. Given these values, equation (12) determines the value of (s d) for every constant d.

Every instance of the Truth Schema has the form

('<-> ('true (quote p)) p)

If we apply a Q-substitution s, this becomes

('<-> ('true (s (quote p))) (s p))

By (12) this is

('<-> ('true (quote (s p))) (s p))

which is again an instance of the Truth Schema. Thus if we add the Truth Schema to our first-order system, Q-substitution will still map proofs to proofs. It remains to consider the Reflection Schema.

If we add the Truth Schema to our proof system, the Reflection Schema must have instances that describe proofs involving the Truth Schema. For example, there should be an instance of the Reflection Schema which says that for all x, the sentence

```
13
 ('<-> ('true (''white (quote x)))
       ('white x)
 )
```

is an instance of the Truth Schema. Then if we let x = "coal", we can infer that

```
 ('<-> ('true (''white (quote 'coal)))
       ('white 'coal)
 )
```

is an instance of the Truth Schema. Since (quote 'coal) = ''coal, this is equal to

```
 ('<-> ('true (''white ''coal))
       ('white 'coal)
 )
```

This term denotes the following sentence:

```
(<-> (true ('white 'coal))
     (white coal)
)
```

It is clearly an instance of the Truth Schema.

We must change the definition of a wff schema so that (13) is a wff schema. First, some notation. If t is a term, ("quote"^n t) is formed by prefixing the function letter "quote" to the term t n times. Thus ("quote"^2 "(arabic N)") is "(quote (quote (arabic N)))". A more exact definition of ("quote"^n t) is: ("quote"^0 t) is t, and ("quote"^n+1 t) is ('quote ("quote"^n t)). Obviously if term t denotes expression e, then ("quote"^n t) denotes (quote^n e). We extend the function "quote" to handle individual symbols as well as terms. If l is a symbol, (quote l) is l with a quotation mark prefixed. So for the connective "&" we have (quote "&") = "'&", and (quote^2 "&") = "''&". Finally, we introduce a new notation for expressions of the object language. If F is an n-adic functor and el ... en are expressions, <F el ... en> is the expression with functor F and arguments el .. en.

The quote name of a wff includes the quote name of every term in that wff. A wff schema is like the quote name of a wff except that variables can appear instead of the quote names of terms. To include (13) as a wff schema, we must generalize this definition. A wff schema is like the quote name of a wff except that expressions of the form ("quote"^n v), where v is a variable, can appear instead of the quote names of terms. We must generalize the definition of a proof schema in the same way, following our observation that a proof schema is to the print name of a proof as a wff schema is to the quote name of a wff. A proof schema is like the print name of a proof except that expressions of the form ("quote"^n v) can appear instead of the quote names of terms. This definition is not completely precise, but it is clear enough to support a convincing proof.

The typical instance of a proof schema is an instance formed by binding its free variables to distinct new constants. If v is a variable and e is an expression, (new v e) is a quoteless constant that does not occur in e (including quoted occurrences). If x is not equal to y, (new x e) is not equal to (new y e). Let p be a proof schema. The typical instance of p is the denotation of p in the environment that binds each variable v to (new v p). I write (den t e) for the denotation of term t in environment e. An environment is just a function from variables to values, so the environment that binds each v to (new v p) is (lambda v. (new v p)). If (typical p) is the typical instance of proof schema p, we have

(typical p) = (den p (lambda v. (new v p)))

Let s be a Q-substitution, and (lambda v. (f v)) an environment that binds each variable v to a closed term (f v). We apply a substitution s to an environment by applying s to each binding in that environment. That is,

14 (s (lambda v. (f v))) = (lambda v. (s (f v)))

As stated in 3.2.2, we must show that if the typical instance of a proof schema is a proof, every instance is a proof. The following lemma is the basis of the argument.

Lemma 1. If p is a proof schema, E binds variables to closed terms, and s is a Q-substitution, then (s (den p E)) = (den (s p) (s E)).

Proof. By induction on sub-expressions of p. It is clear from the definition of the print names of proofs that they contain only two kinds of symbols: quoted function letters and proof-building function letters. A proof schema is like the print name of a proof except that it can contain terms of the form ("quote"^n v), where v is a variable. So a proof schema can contain four kinds of symbols: quoted function letters, proof-building function letters, variables, and the function letter "quote". For the base case, we consider a sub-expression of p that has no sub-expressions of its own. It must be either a quoted constant or a variable. If it is a quoted constant, say (quote c) for some constant c, we have

```
(s (den (quote c) E))
(s c)                                     (quote x) denotes x
(den (quote (s c)) (s E))                 (quote x) denotes x
(den (s (quote c)) (s E))                 s is a Q-substitution
```

If the sub-expression is a variable v, we have

```
(s (den v E))
(den v (s E))                             defn. of (s E)
(den (s v) (s E))                         (s v) = v
```

For the induction step we consider a sub-expression of the form <F al ... an> - the expression with function letter F and arguments al ... an, where n > 0. The function letter F is either a quoted function letter, a proof-building function letter, or the function letter "quote". Let f be the function denoted by the function letter F. We prove

15 (s (f al ... an)) = (f (s al) ... (s an))

If F = (quote G), where G is a functor, we must prove

(s <G al ... an>) = <G (s al) ... (s an)>

Since G is not a constant, this follows at once from the definition of substitution. If F = "quote" we must prove

(s (quote e)) = (quote (s e))

which is true because s is a Q-substitution. Suppose F is the proof-building function letter for the proof rule R. We must prove that applying a substitution s to the proof with rule R, sub-proofs Pl ... Pn, and conclusion c yields the proof with rule R, sub-proofs (s Pl) ... (s Pn), and conclusion (s c). This is true by definition - substitution into proofs is defined component-wise.

With equation (15) we can easily do the induction step.

(s (den <F al ... an> E))	
(s (f (den al E) ... (den an E)))	defn. of denotation
(f (s (den al E)) ... (s (den an E)))	(15)
(f (den (s al) (s E)) ... (den (s an) (s E)))	induction hyp.
(den <F (s al) ... (s an)> (s E))	defn. of denotation
(den (s <F al ... an>) (s E))	F not a constant

This completes the proof of Lemma 1.

Lemma 2. For every proof schema p and Q-substitution s, there is a Q-substitution s' such that (typical (s p)) = (s' (typical p)).

Proof: The substitution s' is like s except that it replaces the new constants of p with the new constants of (s p). If c is a quoteless constant in p,

(s' c) = (s c)

If v is a free variable in p,

(s' (new v p)) = (new v (s p))

These requirements are consistent, because the (new v p)'s are all distinct from each other and from every constant in p. Since (s' c) = (s c) for each constant c in p, (s' p) = (s p). Then

```
(s'(typical p))
(s' (den p (lambda v. (new v p))))          defn. of typical
(den (s' p) (s' (lambda v. (new v p))))     Lemma 1
(den (s' p) (lambda v. (s' (new v p))))     (14)
(den (s' p) (lambda v. (new v (s p))))      defn. of s'
(den (s p) (lambda v. (new v (s p))))       (s' p) = (s p)
(typical (s p))                             defn. of typical
```

This completes the proof.

An instance of Reflection contains a proof schema, which has a typical instance. This typical instance is itself a proof and may use Reflection. We define the depth of nesting of Reflection in a proof in the obvious way. The depth of nesting is 0 in a proof that does not use Reflection. Suppose P is a proof containing Reflection steps, and P1 ... Pn are the typical instances of the proof schemas appearing in these Reflection steps. Then the depth of P is one plus the maximum depth of P1 ... Pn. Now we prove

Lemma 3. If P is a proof, and s is a Q-substitution, then (s P) is a proof.

Proof: By induction on the depth of nesting of Reflection. Suppose the depth of proof P is n, and the theorem holds for all proofs of depth less than n. We have already shown that the non-Reflection steps in (s P) must be correct. Consider a Reflection step in P; it contains a proof schema p. (typical p) is a proof (otherwise P would not be a proof), and its depth is less than n. The corresponding Reflection step in (s P) has the proof schema (s p) and the typical instance (s' (typical p)), which is a correct proof by induction hypothesis. Therefore every Reflection step in (s P) is correct, and (s P) is a proof.

We need one more lemma.

Lemma 4. If p is a proof schema, and E is an environment that binds variables to closed terms, there is a Q-substitution s such that (den p E) = (s (typical p)). That is, every instance of p can be formed by substituting into the typical instance.

Proof: Suppose the environment E binds each variable v to a closed term (f v). For every variable v, let

(s (new v p)) = (f v)

For all other x, (s x) = x. This is a good definition, because if v is not equal to w, (new v p) is not equal to (new w p). (s p) = p, since none of the (new v p)'s occur in p. Then

```
(s (typical p))
(s (den p (lambda v. (new v p)))       defn. of typical
(den (s p) (s (lambda v. (new v p)))) Lemma 1.
(den (s p) (lambda v. (s (new v p)))) (14)
(den (s p) (lambda v. (f v)))         defn. of s
(den p (lambda v. (f v)))             (s p) = p
(den p E)                             defn. of (f v)
```

This completes the proof.

Now we can easily prove

Theorem 1. If p is a proof schema, and the typical instance of p is a correct proof, all instances of p are correct proofs.

Proof. The typical instance of p is a proof P. If P' is any instance of p, by Lemma 4 there is a substitution s such that (s P) = P', so by Lemma 3 P' is a proof. Therefore every instance of p is a proof.

5.2 The Truth Schema Holds for Grounded Sentences

Intuitively, a sentence is grounded if we can find its truth value by the obvious rules without entering an infinite recursion. As a first step in formalizing this notion, let us recall the definition of "infinite recursion" for programs. We define the notion of a recursion of depth n by induction. If the execution of a program does not involve any recursive calls, it is of depth 0. If it involves recursive calls of depths nl ...

nk, its depth is 1 + (maximum nl ... nk). The execution involves infinite recursion if it is not of depth n for any integer n.

Suppose we try to extend this definition to truth value assignments. Consider the rule for assigning a truth value to (all x (P x)). It checks the truth value of (P x) for each choice of x in the domain of the model. Suppose that each of these truth values can be found with a finite depth of recursion. Then certainly there is no infinite regress involved in finding the truth value of (all x (P x)). Yet we cannot define the depth of recursion of (all x (P x)) to be the maximum depth of (P x) for any binding of x. It may be that the model contains infinitely many values for x, and the depth of (P x) can be made arbitrarily large by suitable choice of x. In that case the maximum is not defined.

This problem arises because a set of integers need not have an upper bound in the integers. We can solve it by using infinite ordinals to measure the depth of recursion. Every set of ordinals has an upper bound in the ordinals, so we can define the depth of recursion of (all x (P x)) to be the maximum depth of (P x) for any x in the domain of the model.

A partial model of a language assigns a denotation to every function and predicate letter except "true". This leaves us free to determine the truth value of (true p) by checking the truth value of p. If p is a wff and e is an environment, the pair [p e] is called a closure. Given a partial model, we can attempt to find the truth value of a closure [p e] by recursively applying the truth value rules. If the recursion terminates at depth n with truth value v, the closure has truth value v at level n. If the recursion never terminates, the closure has no level and no truth value - it is ungrounded. We define the truth value and the level of [p e] by the following rules. (If e is an environment, e(v/x) is the environment like e except that the variable v is bound to x.)

If p is an atomic wff whose predicate is not "true", [p e] has the usual truth value at level 0.

If [p e] has truth value T at level n, [~p e] has truth value F at level n+1.

If [p e] has truth value F at level n, ~p has truth value T at level n+1.

If [p e] and [q e] have truth value T at levels n1 and n2, then [p & q e] has truth value T at level (maximum n1 n2) + 1.

If either [p e] or [q e] has truth value F at level n, then [p & q e] has truth value F and level n+1.

If for every x, [p e(v/x)] has truth value T at level f(x), then [(all v p) e] has truth value T, and its level is the least ordinal greater than every f(x).

If for some x, [p e(v/x)] has truth value F at level n, then [(all v p) e] has truth value F at level n+1.

If the denotation of term t in environment e is sentence p, and p has truth value T at level n, then [('true t) e] has truth value T at level n+1. Likewise if p has truth value F.

It is easy to see that for wffs that do not mention the predicate "true", these rules reduce to the ordinary Tarskian rules for truth assignment.

We say that a closure [p e] is grounded if it has a truth value at some level. We can easily see that p = ('~ ('true 'p)) has no truth value any level. For if it has a truth value there is a least ordinal at which it has a truth value, and by the rules for assignment to ~q and (true x), the sentence denoted by p must have a truth value at a lower level. The sentence denoted by p is of course the sentence we started with, so this contradicts the assumption that we have found the least ordinal at which p has a truth value.

We must check that the rules assign unique truth values.
Lemma 4. Every closure has at most one truth value.
Proof: It suffices to prove that for every ordinal n, no closure has two truth values at levels less than or equal to n. The argument is by induction on n. The case n = 0 follows at once from the first clause of the definition. For n > 0, consider a closure of the form [p & q e]. Assume the theorem is false, and [p & q e] has truth values T and F at ordinals less than or equal to n. Then [p e] and [q e] have truth value T at ordinals less than n, and either [p e] or [q e] has truth value F at some ordinal less than n. So either [p e] or [q e] has two truth values at ordinals n1 and n2 less than n. Then the greater of n1 and n2 is a counter-example to the induction hypothesis. So our assumption is false - [p & q e] does not have truth values T and F at ordinals less than or equal to n. The remaining cases are similar.

Given a partial model M, we can construct an ordinary first-order model M' by setting the extension of the predicate "true" to the set of all sentences that have truth value T at some level.
Lemma 5. If [p e] has truth value T (or F) at level n, its truth value in M' is T (or F).
Proof: By induction on the structure of wffs. Consider an atomic wff whose predicate is not "true". Then by the first clause of the definition, if it has truth value T (or F) at some ordinal, it has truth value T (or F) in M'.

Consider next a closure of the form [('true x) e]. If it has truth value T at some ordinal, the denotation of x in environment e has truth value T at some ordinal, so the denotation of x in environment e is in the extension of "true" in M'. Hence [('true x) e] has truth value T in M'. By the same argument, if [('true x) e] has truth value F at any ordinal, it is assigned F in M'.

Next consider the conjunction p & q. If it has truth value T at some ordinal, p and q are assigned T at some ordinals, so by induction hypothesis they are assigned T in M'. Hence p & q is assigned T in M'. If p & q has truth value F at some ordinal, either p or q has truth value F at some ordinal, so by induction hypothesis either p or q is assigned F in M'. Therefore p & q is assigned F in M'. The remaining cases are handled in the same way.

Lemma 6. If p is grounded, the truth value of p in M' is the truth value of ('true (quote p)) in M'.

Proof: Since p is grounded, it has a truth value V in M, and by Lemma 5 its truth value is V in M'. The truth value of ('true (quote p)) in M is also V, by definition. Then the truth value of ('true (quote p)) in M' is also V.

This at once entails

Theorem 2. For any partial model M, if p is a grounded sentence, the sentence

('<-> ('true (quote p)) p)

is true in M'.

We have shown that, given the extensions of the other symbols in our language, we can choose the extension of the predicate "true" so that the Truth Schema is correct for every grounded sentence. Thus the inconsistency of the Truth Schema arises only when dealing with ungrounded sentences.

6. Acknowledgements

I acknowledge my thesis advisor, James Allen, for guidance over the last four years. I thank David Israel for making me see that this theory rests on the implausible but useful assumption that all agents have the same knowledge representation language. Without the work of Don Perlis on theories that contain their own truth predicate, I would have no answer to the objection that self-describing languages are inconsistent.

REFERENCES

[1] Allen, James.
A General Model of Action and Time.
Technical Report, Department of Computer Science, University of Rochester, September, 1981.

[2] Creary, L. G.
Propositional Attitudes: Fregean Representation and Simulative Reasoning.
IJCAI-6:176-181, 1979.

[3] Fodor, J.A.
Representations.
Bradford Books, Montgomery, Massachusetts, 1982, chapter Three Cheers for Propositional Attitudes.

[4] Haas, Andrew.
Planning Mental Actions.
PhD thesis, Department of Computer Science, University of Rochester, 1982.

[5] Haas, Andrew.
What Robots Do and What Robots Can Do.
1983.

[6] Hintikka, J.
Semantics for Propositional Attitudes.
In L. Linksy (editor), *Reference and Modality*, pages 145-167. Oxford University Press, London, 1971.

[7] Kaplan, D.
Quantifying In.
In L. Linsky (editor), *Reference and Modality*, pages 112-144. Oxford University Press, London, 1971.

[8] Konolige, K.
A First-Order Formalization of Knowledge and Action for a Multi-Agent Planning System.
Technical Report 232, SRI International, 1980.

[9] Kripke, Saul.
Outline of a Theory of Truth.
Journal of Philosophy 72(13), 1975.

[10] Lycan, William G.
Towards a Homuncular Theory of Believing.

Cognition and Brain Theory 4(2):139-157, 1981.

[11] McCarthy, John.
Machine Intelligence 9.
Halsted Press, New York, 1979, pages 120-147chapter First-Order Theories of Individual Concepts and Propositions.

[12] McDermott, Drew.
A Temporal Logic for Reasoning About Processes and Plans.
Technical Report 196, Department of Computer Science, Yale University, March, 1981.

[13] Moore, Robert.
Reasoning About Knowledge and Action.
Technical Report 191, SRI International, 1980.

[14] Moore, Robert, and Hendrix, Gary.
Computational Models of Belief and Semantics of Belief Sentences.
Technical Report 187, SRI International, 1979.

[15] Perlis, Donald.
Truth, Syntax and Reason.
PhD thesis, Department of Computer Science, University of Rochester, 1980.

[16] Vilain, Marc.
A System for Reasoning About Time.
In Proceeedings of the 1982 National Conference on Artificial Intelligence, pages 197-201. AAAI, 1981.

Official Distribution List

Contract N00014-77-C-0378

	Copies
Defense Documentation Center Cameron Station Alexandria, VA 22314	12
Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	2
Office of Naval Research Code 200 Arlington, VA 22217	1
Office of Naval Research Code 455 Arlington, VA 22217	1
Office of Naval Research Code 458 Arlington, VA 22217	1
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, MA 02210	1
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, IL 60605	1
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, CA 91106	1
Naval Research Laboratory Technical Information Division Code 2627 Washington, D.C. 20380	6

cont'd.

Naval Ocean Systems Center 1
Advanced Software Technology Division
Code 5200
San Diego, CA 92152

Dr. A. L. Slafkosky 1
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, D.C. 20380

Mr. E. H. Gleissner 1
Naval Ship Research & Development Ctr.
Computation & Mathematics Dept.
Bethesda, MD 20084

Capt. Grace M. Hopper, USNR 1
Naval Data Automation Command
Code 00H
Washington Navy Yard
Washington, D.C. 20374

Mr. Paul M. Robinson, Jr. 1
NAVDAC 33
Washington Navy Yard
Washington, D.C. 20374

Advanced Research Projects Agency 1
Information Processing Techniques
1400 Wilson Boulevard
Arlington, VA 22209

Capt. Richard L. Martin, USN 1
507 Breezy Point Crescent
Norfolk, VA 23511

Director, National Security Agency 1
Attn: R54, Mr. Page
Fort G.G. Meade, MD 20755

Director, National Security Agency 1
Attn: R54, Mr. Glick
Fort G.G. Meade, MD 20755

Major James R. Kreer 1
Chief, Information Sciences
Dept. of the Air Force
Air Force Office of Scientific Research
European Office of Aerospace
Research & Development
Box 14
FPO New York 09510

cont'd.

Mr. Fred M. Griffee Technical Advisor C3 Division Marine Corps Development & Education Command Quantico, VA 22134	1